METHODS & DESIGNS

A strategy for microcomputer-controlled measurement of responses in the behavioral laboratory

RAYMOND F. GENOVESE

Walter Reed Army Institute of Research, Washington, D.C.

Problems associated with microcomputer measurement of responses in operant conditioning experiments are discussed. A general strategy that addresses these problems is presented. Although implementation of the strategy requires assembly language programming, the procedures can be used in conjunction with most high-level programming languages. The method ensures that all responses are detected, prevents detection of multiple responses during switch contact bounce, and indicates when responses are being processed at excessive intervals following their occurrences.

In recent years the popularity of computers in the operant behavior laboratory has grown substantially. Computers are generally more reliable and flexible than the bulky array of electromechanical devices that they have replaced. On the other hand, generating software to control the experimental parameters can cause problems. Several integrated systems are available that, in addition to interfacing the computer and hardware, provide a programming language specifically designed to facilitate the generation of control programs useful in operant conditioning experiments. The most notable and reliable of these systems is SKED (Snapper, Kadden, & Inglis, 1982). More recently, the OPN system also has gained popularity (Emmett-Oglesby, Spencer, & Arnoult, 1982; Spencer & Emmett-Oglesby, 1985).

Alternatively, many experimenters assemble other microcomputer control systems and develop software to control the experimental parameters. Relatively inexpensive designs for constructing suitable hardware interfaces for the Apple II microcomputer were developed by Jenkins (1982) and Gordon, Foree, and Eckerman (1983). Functionally similar interface designs were developed for the VIC-20 (Jackson & O'Dell, 1985) and Commodore-64 (O'Dell & Jackson, 1986) microcomputers. Two systems

using the AIM-65 microcomputer (Carroll, Santi, & Rudell, 1981; Rayfield & Carney, 1981) include assembly language routines designed to facilitate the development of control programs using the BASIC programming language. The routines described by Rayfield and Carney also have been adapted for use with the Apple II computer (Rayfield, 1982); improvements to these software routines were offered by Rush (1986).

The task of assembling a reliable, yet economical, microcomputer system for controlling operant conditioning experiments is complex. This article addresses some of the potential problems associated with microcomputer measurement of responses, specifically leverpresses.

Characterization of a Leverpress as a Response

In general, a response lever is a simple switch. For a microcomputer to measure responses, the response lever is connected by a hardware interface to memory or port locations that are addressable by the microcomputer. Usually, the state of a single response lever is converted to (i.e., represented by) a single bit of the memory or port location. (For a more thorough description of this concept, see Ciarcia, 1986a, 1986b.)

It is convenient to distinguish three components of a leverpress: (1) the transition from an open switch state to a closed switch state, (2) the period of time during which the switch is closed, and (3) the transition from a closed switch state to an open switch state. For the procedures described here, a response is operationally defined as the transition from an open to a closed switch state. This definition of a response is typically, but not necessarily, used in operant conditioning experiments. A single leverpress can, however, produce multiple switch closures, referred to as switch contact bounce. A microcomputer can easily examine an input port frequently

The author thanks T. Elsmore, A. Silberberg, and J. Witkin for helpful comments on the manuscript. In conducting the research described in this report, the investigator adhered to the *Guide for the Care and Use of Laboratory Animals*, as promulgated by the Committee on Care and Use of Laboratory Animals of the Institute of Laboratory Animal Resources, National Research Council. The views of the author do not purport to reflect the position of the Department of the Army or the Department of Defense (para 4-3, AR 360-5).

Requests for reprints should be addressed to R. F. Genovese, Department of Medical Neurosciences, Division of Neuropsychiatry, Walter Reed Army Institute of Research, Washington, DC 20307-5100.

enough to detect each switch closure. Therefore, one problem associated with microcomputer measurement of responses is measuring single leverpresses as multiple responses when switch contact bounce occurs.

Measurement of Switch Contact Bounce

To investigate the extent of switch contact bounce, a Commodore-64 microcomputer was used to measure the characteristics of two rodent response levers commonly used in operant conditioning experiments. Lever A had not previously been used, and Lever B had been used extensively. The Commodore-64 microcomputer includes an input port and a counter/timer with a resolution equal to the system clock period of 0.978 μ sec. The normally closed contacts of the response levers were connected directly to bit 4 of port A of CIA #1, and the common contacts were connected to system ground. (Hardware and programming details of the input port and counter/timer were presented elsewhere [DeJong, 1984, pp. 125-256; Szepesi, 1986].) The duration of the open and closed positions of the switches were measured while 25 leverpresses were made as rapidly as possible. This procedure was used to imitate leverpressing during very fast response rates that are particularly demanding to measure.

Table 1 (top) summarizes the measurements. Both levers consistently produced switch contact bounce during the closure cycle. That is, at the onset of each leverpress, between one and three switch closure and release cycles occurred within 1 msec. The bounce period was followed by two substantially longer periods during which the switch was closed and then open. Neither lever produced switch contact bounce during the release phase of the leverpress. Note that different types of switches exhibit significantly different characteristics. For example, using the same measurement procedure with a membrane-type switch (i.e., the "fire" button on a joystick controller), contact bounce duration frequently exceeds several milliseconds and occurs during both the closure and release phases of the press.

Eliminating Switch Contact Bounce

Software techniques to solve switch contact bounce problems have been suggested by Rayfield and Carney (1981). Since microcomputer processing time is often at a pre-

mium, it may be more practical to implement switch contact debouncing in hardware (Markus, 1982, pp. 21-24). A simple debouncing circuit can be constructed by connecting the switch contacts to a pair of coupled NAND gates. A schematic for this type of circuit was presented by Mimms (1982, p. 41), and similar designs were presented by Jackson and O'Dell (1985) and O'Dell and Jackson (1986). The switching characteristics of the two levers were examined again, after connection to the NAND gate circuit; these measurements also appear in Table 1 (bottom). Switch contact bounce was completely eliminated, while the other switching characteristics were not substantially altered.

The NAND gate debouncing circuit effectively conditions the leverpress signal, whereas other methods of processing the leverpress signal can, under certain circumstances, cause responses to go undetected. For example, leverpresses can be processed through a pulse former. When a leverpress occurs, a signal of a predetermined duration is sent to the input port. Circuit designs of this type were presented by Lancaster (1982, pp. 180-188). Although this method is somewhat more complicated than the NAND gate design, it is also effective if the pulse length is not of a duration that masks the detection of subsequent leverpresses.

Another method of processing leverpresses that can interfere with the accurate detection of responses is latching (e.g., Falcone, Warren, & Rosellini, 1984). Essentially this circuitry debounces a switch input and maintains a signal to the input port until an additional signal, from the microcomputer, clears the latch. If the frequency of examining the input port and clearing the latch exceeds the rate of responding, the technique is effective. If, on the other hand, responses occur faster than the latch is examined and reset, responses go undetected. Additionally, since the response signal to the input port is maintained until the latch is cleared, there is no limit to the duration between the occurrence of a response and its detection.

Failure to Detect Responses

Failure to detect responses because controlling software polls the response manipulanda at a slower frequency than the rate of responding frequently occurs when the soft-

Table 1
Switching Characteristics of Two Rodent Operant Levers,
Without and With Hardware Debouncing

Lever	Bounce Period	Closed Period	Open Period
Without Debounce Circuit			
A	.121 (.068-.199)	51.042 (26.987- 68.934)	73.923 (52.121-102.258)
B	.098 (.077-.173)	58.413 (20.602-186.991)	60.817 (24.450- 98.336)
With Debounce Circuit			
A	0	44.123 (27.763- 68.722)	66.621 (41.130-177.896)
B	0	52.599 (38.722-182.528)	53.779 (33.397- 84.475)

Note—All times are in milliseconds. Each entry represents the mean and range of 25 leverpresses. "Bounce period" refers to the time at the onset of leverpresses during which several rapid switch closure and release cycles were detected.

ware is written in high-level interpreted languages, such as BASIC. Furthermore, the problem is exacerbated when a single microcomputer is used to control several experiments (i.e., concurrent multiple inputs). Certain microcomputer systems that include assembly language poll routines designed to be accessed from BASIC programs do not necessarily prevent responses from going undetected (Carroll, Santi, & Rudell, 1981; Rayfield, 1982; Rayfield & Carney, 1981). Although the frequency of executing the assembly language poll routines may exceed the rate of responding, the frequency of accessing these routines from a BASIC program may not. For example, the assembly language poll routine presented by Rayfield and Carney cannot pass a detected response to the BASIC interpreter if a previous response is waiting to be accessed by the BASIC program. The improvements to Rayfield and Carney's algorithms presented by Rush (1986) do not address this problem. Rayfield correctly noted that if a compiled BASIC language is used, a much faster response rate can be measured. In practice, if the BASIC commands (compiled or not) that are executed following the detection of a response exceed the time during which two or more responses occur, responses go undetected.

A Strategy for Response Measurement

One strategy for effectively measuring responses in operant conditioning experiments is to provide an input poll frequency that is faster than the rate of responding but slower than the duration of switch contact bounce. Some switches have contact bounce durations that make it impractical to meet this requirement. Therefore, hardware switch debouncing should be used. Since hardware switch debouncing can be accomplished easily and inexpensively, it is advisable that it always be used in any microcomputer-controlled system for operant conditioning experiments.

An effective strategy should ensure that all responses are detected and processed within a reasonable period following their occurrence. That is, the procedure should include a method for reporting when software controlling the experimental parameters is executing too slowly to adequately process responses. A strategy that meets these requirements is presented in Figure 1. For clarity, the input poll routine described here assumes a single response lever but can be expanded for use with multiple inputs. The strategy assumes that neither the latching nor the pulse-forming types of switch signal processing referred to earlier are used.

The procedure is implemented through the use of fixed-rate interrupts. When an interrupt signal occurs, processing is directed to instructions at a specific location. Following the execution of these instructions, processing returns to the instructions at the point of interruption. This method is well suited to polling the input port since the interrupts occur at regular intervals regardless of the main program execution. Additional routines, which are not

interrupt-driven, can interact with locations used by the input poll routine to detect responses.

Virtually all of the popular microprocessors have an interrupt capability, and many include hardware to generate interrupts at frequencies that can be programmed by the user. Since the execution frequency of the routine that polls the input port should be faster than the rate of responding, the latter feature is very convenient. Additional hardware may be constructed to produce programmable interrupt frequencies for microcomputers that do not already have this feature. Based on the data presented in Table 1, an interrupt frequency of 10 msec is adequate.

Algorithm for an Input Poll Routine

The input poll routine makes particular use of three memory locations: (1) a last-state flag representing the position of the response lever during the last interrupt, (2) a response counter that is incremented when the interrupt routine detects a transition from an open switch state to a closed switch state, and (3) a cycle counter that counts the number of interrupts that have occurred before a response is acknowledged by the main program. The main program detects responses by examining the response counter. If the value of the response counter is 1, a response has occurred, and the main program decrements the counter and processes the response.

The first function of the input routine is to examine the bit of the input port that represents the response lever. If the lever is in the open position, the last-state flag is reset and the response counter is examined. If the value is 0, the cycle counter is cleared and the routine executes additional interrupt functions (e.g., timers), if present, and then returns.

When the response counter contains a value greater than 0, a response has been detected but has not yet been processed by the main program. In this case, the value of the cycle counter is incremented and compared to a constant that determines the maximum interval between the detection of a response and the start of response processing by the main program. In this example, a constant of 10 represents 10 interrupts at 10-msec intervals, or 100 msec. When the cycle counter exceeds the allowable count, an error routine is executed. When the value of the cycle counter is less than the constant, the interrupt routine continues.

If the bit of the input port that represents the response lever indicates that the switch is in the closed position, the last-state flag is examined. When the last-state flag is set, the response has already been detected by the interrupt routine (i.e., the response lever is being held down), and execution continues at the point where the response counter is compared to a value of 0. A valid response has been detected when the last-state flag is reset. In this case, the last-state flag is set and the response counter is incremented. When the value of the response counter is not greater than 1, execution continues at the point

Figure 1. A flowchart describing the interrupt-driven strategy for measuring responses in operant research experiments.

where the cycle counter is incremented. When the value of the response counter is greater than 1, a response has been detected before a previous response has been acknowledged by the main program and an error routine is executed.

Several approaches to error processing can be implemented. Ideally, program execution should be halted when an error is detected, and the main program should be refined to increase execution speed. Subsequently, the error detection routines can increment memory locations to serve as error counters. To provide a regular integrity check, these locations can be examined after each program run.

Summary

The response measurement strategy presented here has three distinctly favorable features. First, by executing the input poll routine at a frequency that exceeds the maximum response rate, all responses are detected. Furthermore, if hardware debouncing is not used, all responses are detected and switch contact bounce problems are over-

come if the input poll routine executes at a frequency exceeding the maximum response rate but not the maximum switch bounce duration. Second, the procedure reports when responses are not being processed quickly enough because other programming functions require excessive amounts of time. Finally, although the interrupt-driven poll routine and the associated response detection routine necessarily must be written in assembly language, they can be utilized by programs written in a variety of higher level languages such as BASIC, C, FORTRAN, and Pascal.

REFERENCES

- CARROLL, M. E., SANTI, P. A., & RUDELL, R. L. (1981). A micro-computer system for control of behavioral experiments. *Pharmacology Biochemistry & Behavior*, *14*, 415-417.
- CIARCIA, S. (1986a). Parallel interfacing: A tutorial discussion: Part 1. Basics. *Byte*, *11*, 85-92.
- CIARCIA, S. (1986b). Parallel interfacing: A tutorial discussion: Part 2. Applications. *Byte*, *11*, 97-105.
- DEJONG, M. L. (1984). *Assembly language programming for the Commodore-64*. Bowie, MD: Prentice-Hall.

- EMMETT-OGLESBY, M. W., SPENCER, D. G., & ARNOULT, D. E. (1982). A TRS-80 based system for the control of behavioral experiments. *Pharmacology Biochemistry & Behavior*, **17**, 583-587.
- FALCONE, R. V., WARREN, D. A., & ROSELLINI, R. A. (1984). Response input interface. *Behavior Research Methods, Instruments, & Computers*, **16**, 285-287.
- GORDON, W. A., FOREE, D., & ECKERMAN, D. A. (1983). Using an Apple II microcomputer for real-time control in a behavioral laboratory. *Behavior Research Methods & Instrumentation*, **15**, 158-166.
- JACKSON, D. E., & O'DELL, J. W. (1985). The VIC-20 computer and an interface system for controlling operant chambers. *Behavior Research Methods, Instruments, & Computers*, **17**, 406-408.
- JENKINS, W. M. (1982). Interfacing the Apple II for the behavioral laboratory. *Behavior Research Methods & Instrumentation*, **14**, 342-347.
- LANCASTER, D. (1982). *TTL cookbook*. Indianapolis, IN: Sams.
- MARKUS, J. (1982). *Digital circuits ready-reference*. New York: McGraw-Hill.
- MIMMS, F. M., III (1982). *Engineer's notebook II*. Ft. Worth, TX: Tandy.
- O'DELL, J. W., & JACKSON, D. E. (1986). The Commodore-64 and an interface system for controlling operant chambers. *Behavior Research Methods, Instruments, & Computers*, **18**, 339-341.
- RAYFIELD, F. (1982). Experimental control and data acquisition with BASIC in the Apple computer. *Behavior Research Methods & Instrumentation*, **14**, 409-411.
- RAYFIELD, F., & CARNEY, J. (1981). Controlling behavior experiments with BASIC on 6502-based microcomputers. *Behavior Research Methods & Instrumentation*, **13**, 735-740.
- RUSH, D. K. (1986). Apple II assembly language routines for digital data acquisition and contingency control. *Behavior Research Methods, Instruments, & Computers*, **18**, 32-35.
- SNAPPER, A. G., KADDEN, R. M., & INGLIS, G. B. (1982). State notation of behavioral procedures. *Behavior Research Methods & Instrumentation*, **14**, 329-342.
- SPENCER, D. G., & EMMETT-OGLESBY, M. W. (1985). Parallel processing strategies in the application of microcomputers to the behavioral laboratory. *Behavior Research Methods, Instruments, & Computers*, **17**, 294-300.
- SZEPESI, Z. (1986). Microsecond timer for the Commodore-64. *The Transactor*, **6**, 35-37.

(Manuscript received June 15, 1987;
revision accepted for publication October 25, 1987.)